(19) JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: **08123700 A**

(43) Date of publication of application: **17 . 05 . 96**

(51) Int. Cl **G06F 9/46**

(21) Application number: **06255403**

(22) Date of filing: **20 . 10 . 94**

(71) Applicant: **FUJITSU LTD**

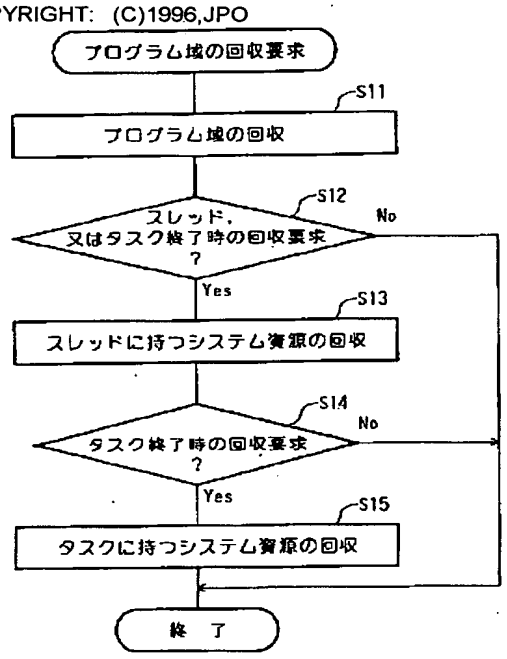
(72) Inventor: **YOKOO SEIKICHI**

(54) **RESOURCE MANAGING METHOD**

(57) Abstract:

PURPOSE: To improve execution performance at the end of a thread and at the time of thread reproduction by recovering only system resources that a program area and the thread have at the end of the thread.

CONSTITUTION: At a request to recover the program area, the program area is recovered (S11) and it is decided whether or not the request to recover the program area is a recovery request at the end of a thread or task (S12; when the request is a dynamic request to recover the program area, the process is ended. When the request is the recovery request at the end of the thread or task, the resources that the thread has are recovered (S13) and it is decided whether or not the recovery request is the recovery request at the end of the task (S14). When the recovery request is the recovery request at the end of the thread, the process is ended. When the request is the recovery request at the end of the task, the resources that the task has are recovered (S15) and the process is ended.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平8－123700

(43)公開日　平成８年(1996)５月17日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　9/46	３４０　Ｆ	7737－5Ｂ		

審査請求　未請求　請求項の数４　ＯＬ　（全　８　頁）

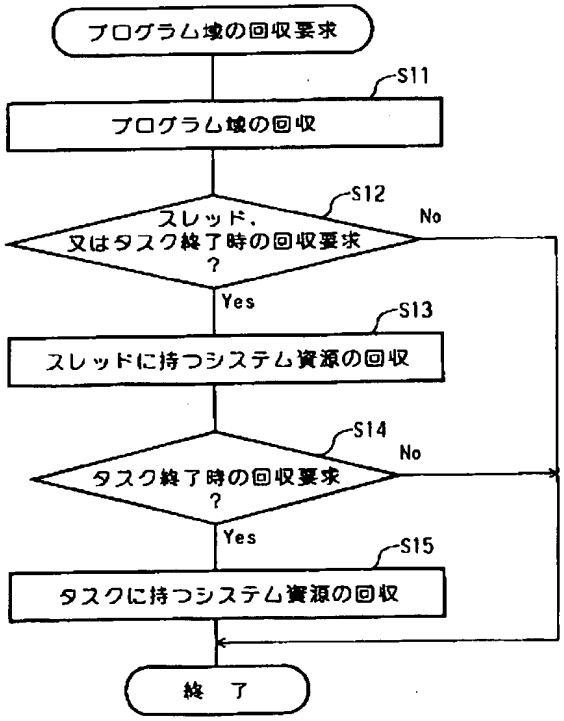
(21)出願番号　特願平6－255403

(22)出願日　平成６年(1994)10月20日

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中1015番地
(72)発明者　横尾　清吉
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内
(74)代理人　弁理士　石田　敬　（外３名）

(54)【発明の名称】　資源管理方法

(57)【要約】
【目的】　資源管理方法において、スレッド終了時とスレッド再生時の実行性能及び、動的なプログラム域の回収要求時とプログラム域の再獲得要求時の実行性能を向上させる。
【構成】　空間内に１又は複数のタスクを有し、タスク内に１又は複数のスレッドを有し、タスク単位でファイル管理制御表を用いてファイル管理を行い、スレッド単位でプログラム域管理制御表を用いてプログラム管理を行う資源管理方法において、スレッド終了時には、プログラム域とそのスレッドに持つシステム資源のみを回収する。また、動的なプログラム回収要求時には、プログラム域のみを回収する。プログラム域の獲得時又はスレッド再生時の実行性能を向上させることが可能となる。



【特許請求の範囲】

【請求項１】　空間内に１又は複数のタスクを有し、前記タスク内に１又は複数のスレッドを有し、前記タスク単位でファイル管理制御表を用いてファイル管理を行い、前記スレッド単位でプログラム域管理制御表を用いてプログラム管理を行う資源管理方法において、スレッド終了時には、プログラム域とそのスレッドに持つシステム資源のみを回収することを特徴とする資源管理方法。

【請求項２】　前記プログラム域とそのスレッドに持つシステム資源の回収は、前記プログラム域管理制御表が使用していた領域を回収することにより行うことを特徴とする請求項１記載の資源管理方法。

【請求項３】　空間内に１又は複数のタスクを有し、前記タスク内に１又は複数のスレッドを有し、前記タスク単位でファイル管理制御表を用いてファイル管理を行い、前記スレッド単位でプログラム域管理制御表を用いてプログラム管理を行う資源管理方法において、動的なプログラム回収要求時には、プログラム域のみを回収することを特徴とする資源管理方法。

【請求項４】　前記プログラム域の回収は、プログラム域として使っていた領域を回収し、前記プログラム域管理制御表で管理しているプログラム域の開始アドレスをクリアすることにより行われることを特徴とする請求項１記載の資源管理方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、コンピュータの資源管理に関するものである。

〔技術の背景〕始めに、プログラムの実行体について図３を用いて説明する。図に示すように、プログラムの実行体は、空間１、タスク２、スレッド３から構成される。

【０００２】空間１は、システムコールなどの定められたインタフェースに基づいて、空間外から制御可能な唯一の単位である。空間１は、それぞれが１つの独立したアドレス空間で構成され、空間１の内部では、内部実行体であるタスク２及びスレッド３が動作してプログラムを実行する。空間間では、インタフェースを介さずに互いに影響を及ぼすことはできない。したがって、セキュリティの上で、最も強固な実行環境である。

【０００３】タスク２は、空間１内に１つの独立したプログラム実効環境（例えば、Ｃ言語では main 関数を持つプログラムの実行環境）を提供する。空間１内では、複数のタスク２が存在してプログラムを多重に実行することが可能である。タスク２は、特に、通信アプリケーションプログラムの実行環境を構築する際に有効なものである。タスクの独立性の高いプログラム実行環境により、１つの空間内でアプリケーションプログラムを多重に実行することができるため、空間資源の削減がはかれる。

【０００４】スレッド３は、ＣＰＵの実行権を得てプログラムの命令列を実行する最も基本的な制御単位である。個々のタスク２は、その内部に複数のスレッド３を以てプログラムを並列処理することが可能である。各スレッド３には、スレッドの持つ実行優先度にしたがって、ＣＰＵの実行権が時分割で分配される。スレッドがＣＰＵの実行権を得ると、そのスレッドに結び付けられたプログラムの命令コードが実行される。実行スレッドの切り換えに際しては、切り換え前の実行環境は退避され、再実行時には退避された実行環境が復元される。

【０００５】スレッド３は、特にＧＳ（ＯＳのサーバ）の並列処理を実現する際に有効である。ＧＳでは、クライアントからの処理要求を高速に処理するために、個々のスレッドが、クライアントからのそれぞれの処理要求を処理するように使用される。以上のようなプログラム実行体の構造化により、元々プログラム実行体が合わせ持っていた諸資源、諸属性は、空間資源４、タスク資源５、スレッド資源６に分解されている（これらの資源については後述する。）。

【０００６】このようなプログラム実行体の構造化により、プログラムの用途ごとに最適な実行環境と実行形態を実現している。ここで、空間資源４、タスク資源５、スレッド資源６について説明をする。空間１は、空間内の全タスク２により共用される空間資源４を持つ、このような資源には、システム内で空間を一意に識別するプロセスｉｄ、空間の権限を示す利用者ｉｄ、グループｉｄなどがある。これらの空間資源４は、そもそも空間固有の性質を持ち、下位の実行体（タスク２又はスレッド３）に割りつけられ得ない資源、プログラムによって変更される可能性が少なくタスク２による共有に問題のない資源等である。（逆に、空間１が複数タスク２を持って動作する場合には、各タスク内で動作するプログラムは、他のタスクに影響を及ぼすような空間資源の変更を制限される。）

以下は、代表的な空間資源４の一覧である。

１．プロセスｉｄ
２．親プロセスｉｄ
３．プロセスグループｉｄ
４．セション
５．実利用者ｉｄ、実グループｉｄ、実効利用者ｉｄ、実効グループｉｄ、退避利用者ｉｄ、退避グループｉｄ
６．カレントワーキングディレクトリ
７．ルートディレクトリ
８．ファイルモード生成マスク
９．外部変数environ から指される環境変数
１０．ロードモジュールライブラリ
１１．事前オープンファイル

ただし、環境変数については、タスクが自身の外部変数 environ の値を変更し、タスク固有の環境変数域を指す

ように変更することができる。

【０００７】個々のタスク２は、それぞれがプログラムが使用する外部変数やファイルのオープン環境などのタスク資源を持ち、他タスクとの資源の競合を起こさないプログラム実行環境を提供する。以下は、代表的なタスク資源５の一覧である。

１．タスクｉｄ
２．プログラムの外部変数域
３．ファイルディスクリプタ、Ｉ／Ｏバッファ
４．動的獲得領域

スレッド３は、以下の代表的なスレッド資源だけを持ち、軽量な実行制御の体を実現している。

１．スレッドｉｄ
２．ＰＳＷや汎用レジスタなどのＣＰＵ資源
３．プログラムの静的局所変数域
４．シクナルのマスタとシグナルの応答動作

【０００８】

【従来の技術】従来の資源管理方法においては、プログラム域とこれを管理するシステム資源を一体と考え、プログラム域を回収するときは、システム資源も回収の対象としていた。ところが、技術の進展に伴い、プログラム域が日々増加しているため、これを管理するシステム資源も当然増加してきている。

【０００９】

【発明が解決しようとする課題】このような状況にあっては、シングルタスクが動作する環境においては、スレッドの終了が空間の終了と同一であるため、わざわざスレッドやタスクで管理している資源を回収する必要はないので、実行性能を悪化させることはない。しかしながら、マルチタスクが動作する環境においては、スレッドの終了が空間まで終了するわけではないので、陽に資源を回収する必要がある。したがって、マルチタスクにおいては、回収すべきプログラム域又はこれを管理するシステム資源が増加すると、実行性能を著しく悪化させてしまう。

【００１０】本発明は、資源管理方法において、スレッド終了時とスレッド再生時の実行性能を向上させることを目的とする。更に、本発明は、動的なプログラム域の回収要求時とプログラム域の再獲得要求時の実行性能を向上させることを目的とする。

【００１１】

【課題を解決するための手段】本発明は、上記第１の目的を達成するため、空間内に１又は複数のタスクを有し、前記タスク内に１又は複数のスレッドを有し、前記タスク単位でファイル管理制御表を用いてファイル管理を行い、前記スレッド単位でプログラム域管理制御表を用いてプログラム管理を行う資源管理方法において、スレッド終了時には、プログラム域とそのスレッドに持つシステム資源のみを回収する。

【００１２】前記プログラム域とそのスレッドに持つシステム資源の回収は、プログラム域管理制御表が使用していた領域を回収することにより行うことができる。本発明は、上記第２の目的を達成するため、空間内に１又は複数のタスクを有し、前記タスク内に１又は複数のスレッドを有し、前記タスク単位でファイル管理制御表を用いてファイル管理を行い、前記スレッド単位でプログラム域管理制御表を用いてプログラム管理を行う資源管理方法において、動的なプログラム回収要求時には、プログラム域のみを回収する。

【００１３】なお、動的なプログラム回収要求時とは、前述した空間、タスク、スレッドと言った資源の回収契機による回収要求ではなく、ユーザが陽にプログラム域の回収要求を行うことを言う。前記プログラム域の回収は、プログラム域として使っていた領域を回収し、プログラム域管理制御表で管理しているプログラム域の開始アドレスをクリアすることにより行うことができる。

【００１４】

【作用】本発明では、プログラム域の管理をプログラム域のスコープであるスレッドだけで管理するのではなく、その上位の実行単位であるタスクでも管理することを特長としている。このため、マルチタスクにおいては、スレッド終了時にはスレッドで管理している資源のみを回収し、タスクで管理している資源は回収しないことにより、回収すべき資源の量を減らしてスレッド終了時の実行性能を向上させる。また、スレッドの再生成時には、回収しなかったシステム資源を再利用することにより、スレッド再生成時の実行性能を向上させる。

【００１５】シングルタスクにおいては、動的なプログラム域の回収要求時にはプログラム域のみを回収し、スレッドで管理している資源は回収しないことにより、回収すべき資源の量を減らして動的なプログラム域の回収要求時の実行性能を向上させる。また、プログラム域の再生成には、回収しなかったプログラム域を再利用することにより、プログラム域の再生成時の実行性能を向上させる。

【００１６】

【実施例】始めに、本発明の資源管理方法が適用される各種空間の構造について図４～図６を用いて説明する。空間は、プログラムの実行環境や実行形態の違いから、例えば、バッチと会話処理用の空間、通信アプリケーション空間、ＧＳ空間の３種類に分けられる。

【００１７】図４は、シングルタスクの環境であるバッチと会話処理用の空間の構造を示す。バッチと会話処理用の空間７は、空間内で１つのタスク２と１つのスレッド３が動作する形態で構成される。この空間７で動作するアプリケーションプログラムにとっては、プログラム実行体が空間７、タスク２、スレッド３から構成されていることはあまり重要なことではない。アプリケーションプログラムは、空間７内の全資源を専有して走行する。このように１空間、１タスク、１スレッドが縮退し

てプログラム実行体を形成する形態はプロセスと呼ばれる。

【００１８】バッチと会話処理用の空間７は、個々のアプリケーションプログラムやユーティリティプログラムが実行される度に作成され、そのプログラムの終了とともに削除される。図５は、マルチタスクの環境である通信アプリケーション空間の構造を示す。通信アプリケーション空間８では、１つの空間で複数のタスク２，２が多重に走行する。個々のタスク２ではそれぞれ１つのスレッド１が走行してアプリケーションプログラムを実行する。個々のタスク２で動作するファイルは、タスクの独立性の高いプログラム実行環境下で、空間８内の他タスクの存在を殆ど意識しないで走行することができる。１０は事前オープンファイル環境である。

【００１９】通信アプリケーション空間８は、アプリケーションプログラムが処理を開始する以前に前もって作成され、停止時に削除される。これらの空間８では、アプリケーションプログラムが使用するファイルを空間作成時に事前にオープンしておくことが可能である。これによって、タスクで動作するアプリケーションプログラムは、少ないオーバヘッドでファイルを使用できるようになる。

【００２０】図６は、シングルタスクの環境であるＧＳ空間の構造を示す。ＯＳの各サーバ空間をＧＳ（Global Service）空間と言う。ＧＳ空間９では、タスク２が１つだけ存在し、その中で複数のスレッド３，３が走行する。クライアントからの処理要求は、それぞれが軽量なスレッドで高速に処理される。ＧＳ空間９では、タスク２はあまり重要な役割を果たさない。

【００２１】このＧＳ空間９は、システム始動時又はstartgsユーティリティ実行時に作成され、システム停止時又はstopgsユーティリティ実行時に削除される。本発明は、上記各種空間のうち、通信アプリケーション空間８のようなマルチタスクの環境を必要とするシステムにおいても、バッチと会話処理用の空間７及びＧＳ空間９のようなシングルタスクの環境においても有効である。

【００２２】次に、本実施例におけるプログラム域とシステム資源の関係を図７を用いて説明する。図７において、１１はタスク資源表、１２はファイル管理制御表である。ファイル管理制御表１２はタスク資源であり、ファイル単位にチェーンが続く。ファイル管理制御表１２には、ファイルの排他、ファイルリードに必要な情報等のファイルのオープン環境が記録されている。

【００２３】１３はスレッド資源表、１４はプログラム域管理制御表である。プログラム域管理制御表１４はスレッド資源であり、プログラム単位にチェーンが続く。プログラム域環境制御表１４には、ファイル管理制御表１２をポイントするファイルのオープン環境と、プログラム域１５の開始アドレスとサイズ等のプログラム域管理が記録されている。１５はプログラム域である。

【００２４】次に、プログラム域の回収要求時とプログラム域の再獲得要求時の処理フローを図１及び図２を用いて説明する。図１の処理フローは、プログラム域の回収要求時にスタートする。プログラムの実行体である空間、タスク、スレッドの各実行体が終了する際、各実行体が持つ資源の自動回収が行われる。例えば、スレッドと言う実行体が終了する際、スレッド資源を持つコンポーネントに対して資源の回収要求が行われる。ここで言うスレッド終了時の回収要求とは、この要求のことをいう。ここで、本発明に関するコンポーネントはスレッド資源を持つので、スレッド終了時にシステムからスレッド資源回収の契機がもらえる。このときにスレッド資源を必要に応じて回収する。

【００２５】同様に、タスクと言う実行体が終了する際にも、システムはタスク資源を持つコンポーネントに対して資源の回収要求を行う。この要求がタスク終了時の回収要求である。また、上記の空間、タスク、スレッドといった資源の回収契機による回収要求でなく、ユーザが陽にプログラム域の回収要求を行う動的なプログラム域の回収要求も行われる。具体的には、スーパバイザがユーザ向けにプログラム域を回収する関数を用意しており、ユーザはこの関数を使ってプログラム域の回収をスーパバイザに要求する。主なコンポーネントとしては、シェルがあり、その使用目的は時空間（シェル空間のこと）の仮想領域が有限であることから、資源の有効利用のために利用される。

【００２６】プログラム域の回収要求があると、ステップＳ１１において、プログラム域の回収が行われる。プログラム域の回収とは、プログラム域１５として使っていた領域を回収し、プログラム域管理制御表１４で管理しているプログラム域１５の開始アドレスをクリアすることを言う。次に、ステップＳ１２で、プログラム域の回収要求がスレッド又はタスク終了時の回収要求であるか否かが判定される。もし、前述の動的なプログラム域の回収要求であれば、ステップＳ１１のプログラム域の回収のみで処理を終了する。

【００２７】スレッド又はタスク終了時の回収要求である場合はステップＳ１３へ進み、スレッドに持つ資源の回収を行う。スレッドに持つ資源の回収とは、プログラム域管理制御表１４で使っていた領域を回収することを言う。ステップＳ１４では、回収要求がタスク終了時の回収要求であるか否かが判定される。回収要求がスレッド終了時の回収要求であれば、ステップＳ１１のプログラム域の回収とステップＳ１３のスレッドに持つ資源の回収のみで処理を終了する。

【００２８】タスク終了時の回収要求である場合は、ステップＳ１５へ進み、タスクに持つ資源の回収を行い処理を終了する。タスクに持つ資源の回収とは、ファイル管理制御表１２で使っていた領域を回収し、またこのファイル管理制御表１２が管理していた情報であるファイ

ルのオープン環境の返却を言う。タスク終了時の回収要求の場合は、ステップＳ１１のプログラム域の回収と、ステップＳ１３のスレッドに持つ資源の回収と、ステップＳ１５のタスクに持つ資源の回収が行われる。

【００２９】図２の処理フローは、プログラム域の再獲得要求時にスタートする。プログラム域の再獲得要求があると、ステップＳ２１でスレッドに持つ該当システム資源があるか否かが判定され、ステップＳ２２でタスクに持つ該当システム資源があるか否かが判定される。スレッドに持つ該当システム資源及びタスクに持つ該当システム資源が共になければ、ステップＳ２３でタスクに持つ該当システム資源の獲得が行われ、ステップＳ２４でスレッドに持つシステム資源の獲得が行われ、ステップＳ２５でプログラム域の獲得が行われる。

【００３０】ここで、タスクに持つシステム資源の獲得とは、ファイル管理制御表１２としての領域を獲得し、このファイル管理制御表１２にファイルのオープン環境を構築することを言う。スレッドに持つシステム資源の獲得とは、プログラム域管理制御表１４としての領域を獲得し、このプログラム域管理制御表１４にファイルのオープン環境とて使う資源表をポイントし、プログラム域のサイズを設定することを言う。

【００３１】プログラム域の獲得とは、プログラム域としての領域を獲得し、この領域にプログラムの内容を読み込み、必要ならば再配置等の処理も行い、この獲得したアドレスをプログラム域の開始アドレスとしてプログラム域管理制御表１４に設定することを言う。スレッドに持つ該当システム資源があり、タスクに持つ該当システム資源がない場合は、ステップＳ２４でスレッドに持つシステム資源の獲得が行われ、ステップＳ２５でプログラム域のみの獲得が行われる。

【００３２】また、スレッドに持つ該当システム資源ある場合には、ステップＳ２５でプログラム域のみの獲得が行われる。図８は、資源の回収状態を従来の方法と比較している。この表を見ても分かるように動的なプログラム域の回収要求ではプログラム域だけしか回収しないため、回収すべき資源の量を減少させる。また、同じプログラム域に対する再獲得要求については単にプログラム域の獲得だけで処理が完結する。

【００３３】また、スレッド終了時の回収要求では、プログラム域とスレッドに持つシステム資源だけを回収するため、回収すべき資源の量を減少させる。また、スレッド再生時の再獲得要求については、プログラム域とスレッドに持つシステム資源だけの獲得で処理が完結する。このため、従来と同じ再獲得要求については処理量がすくなくなり、実行性能を向上させることができる。

【００３４】以上説明した資源管理方法は、シングルタスクに関しては、動的なプログラム域の回収要求と、プログラム域の再獲得要求の場合に有効である。また、マルチタスクに関しては、タスクの環境を再利用する場合に有効である。なお、タスクが終了する場合はスレッドに持つ資源、及びタスクに持つ資源を全て回収するので、本発明がタスク単位に実現するアプリケーションの活性保守を妨げることはない。

【００３５】

【発明の効果】以上説明したように、本発明によれば、資源管理方法において、スレッド終了時とスレッド再生時の実行性能を向上させ、更に、動的なプログラム域の回収要求時とプログラム域の再獲得要求時までも実行性能を向上させることができる。

【図面の簡単な説明】

【図１】本発明の実施例のプログラム域の回収要求時の処理フローを示すフローチャート。

【図２】本発明の実施例のプログラム域の再獲得要求時の処理フローを示すフローチャート。

【図３】プログラムの実行体の構成を示す図。

【図４】バッチと会話処理用の空間の構造を示す図。

【図５】通信アプリケーション空間の構造を示す図。

【図６】ＧＳ空間の構造を示す図。

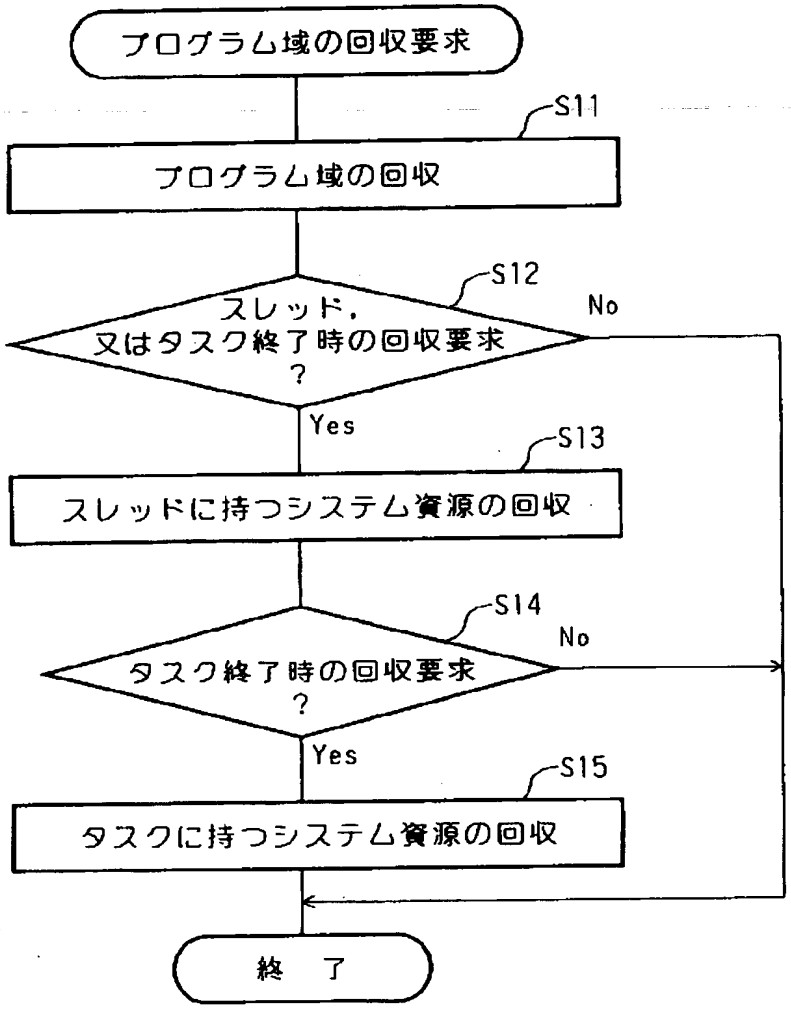
【図７】プログラム域とシステム資源の関係を示す図。

【図８】本発明の実施例による資源の回収状態と、従来例による資源の回収状態を比較して示す図。

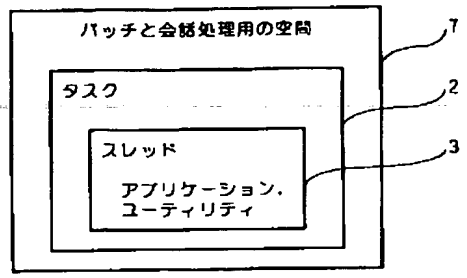
【符号の説明】

１…空間
２…タスク
３…スレッド
４…空間資源
５…タスク資源
６…スレッド資源
７…バッチと会話処理用の空間
８…通信アプリケーション空間
９…ＧＳ空間
１０…事前オープンファイル環境
１１…タスク資源表
１２…ファイル管理制御表
１３…スレッド資源表
１４…プログラム域管理制御表
１５…プログラム域

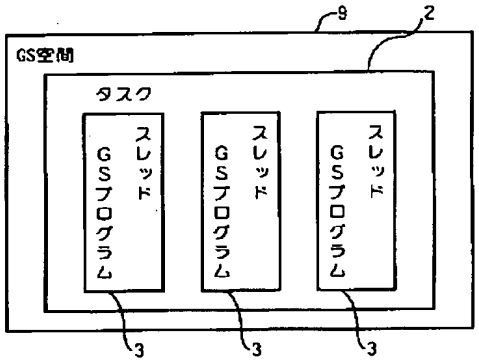
【図１】


プログラム域の回収要求
S11
プログラム域の回収
S12
スレッド，
又はタスク終了時の回収要求
？
No
Yes
S13
スレッドに持つシステム資源の回収
S14
タスク終了時の回収要求
？
No
Yes
S15
タスクに持つシステム資源の回収
終　了

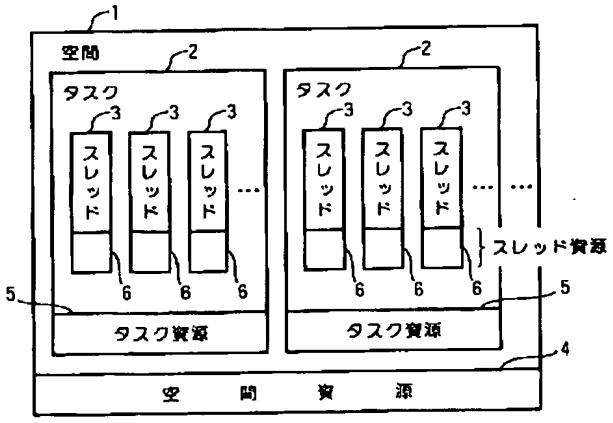
【図４】


バッチと会話処理用の空間
7
タスク
2
スレッド
アプリケーション，
ユーティリティ
3

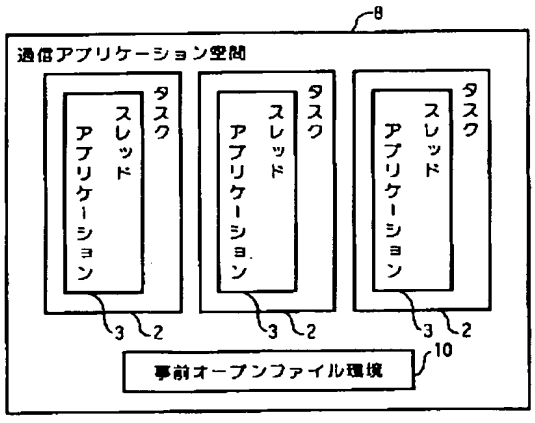
【図６】


9
2
GS空間
タスク
スレッド
GSプログラム
スレッド
GSプログラム
スレッド
GSプログラム
3
3
3

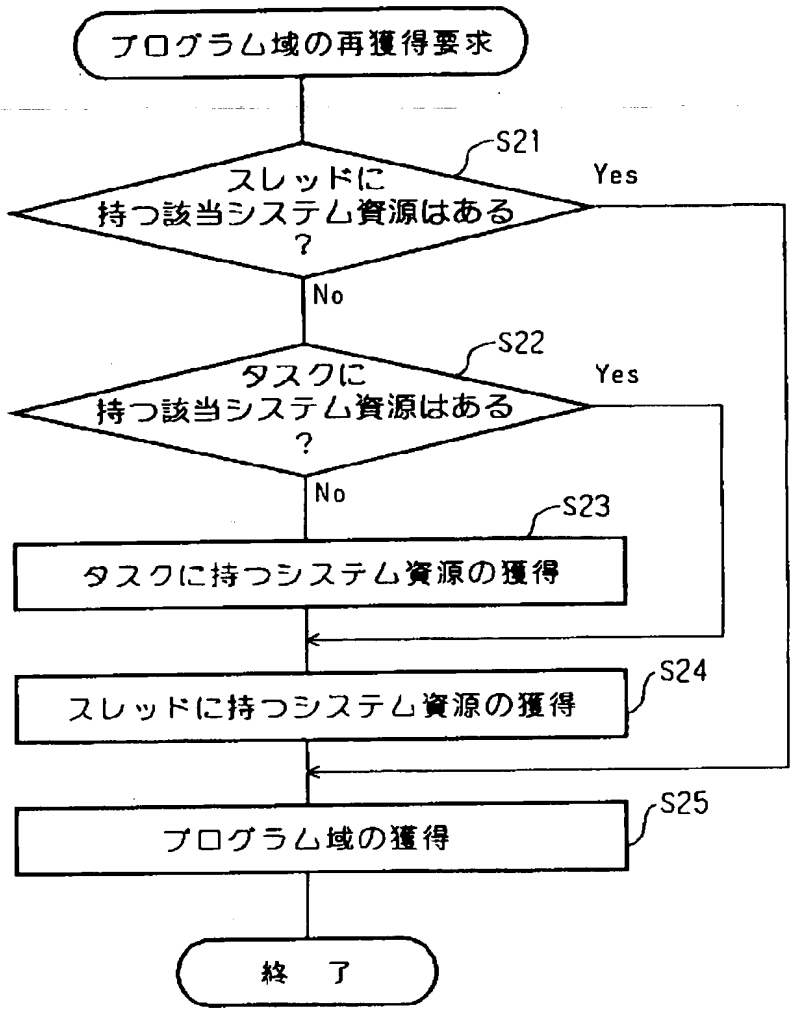
【図３】


1
空間
2
タスク
3
スレッド
6
スレッド資源
5
タスク資源
4
空　間　資　源

【図５】


8
通信アプリケーション空間
タスク
スレッド
アプリケーション
3
2
10
事前オープンファイル環境

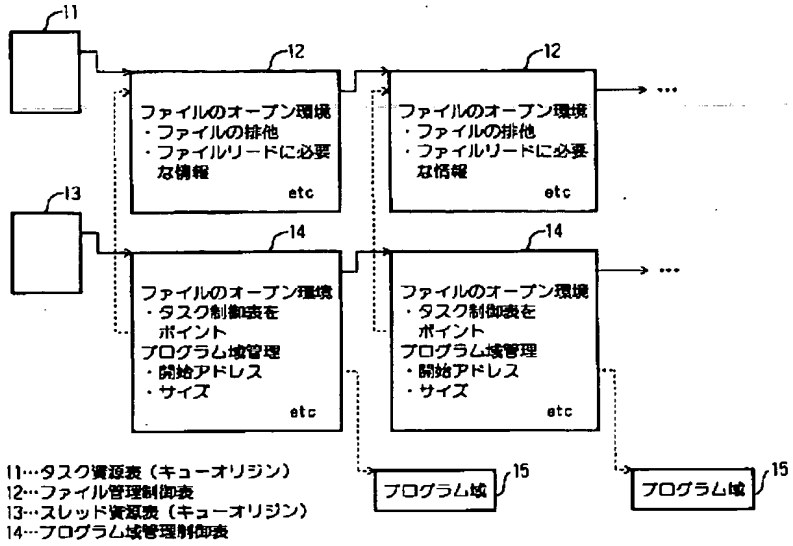
【図２】



【図８】

		動的なプログラム域の回収要求時	スレッド終了時の回収要求	タスク終了時の回収要求
プログラム域	従　来	回収する	回収する	回収する
	新方式	回収する	回収する	回収する
スレッドに持つシステム資源	従　来	回収する	回収する	回収する
	新方式	回収しない	回収する	回収する
タスクに持つシステム資源	従　来	回収する	回収する	回収する
	新方式	回収しない	回収しない	回収する

【図７】


11
12
ファイルのオープン環境
・ファイルの排他
・ファイルリードに必要な情報
etc
12
ファイルのオープン環境
・ファイルの排他
・ファイルリードに必要な情報
etc
13
14
ファイルのオープン環境
・タスク制御表をポイント
プログラム域管理
・開始アドレス
・サイズ
etc
14
ファイルのオープン環境
・タスク制御表をポイント
プログラム域管理
・開始アドレス
・サイズ
etc
プログラム域
15
プログラム域
15
11…タスク資源表（キューオリジン）
12…ファイル管理制御表
13…スレッド資源表（キューオリジン）
14…プログラム域管理制御表